VŠB – Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Computer Science

Individual Professional Practice in the Company

Absolvování individuální odborné praxe

2019

Duy Van Khanh Phan

VŠB - Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Computer Science

Bachelor Thesis Assignment

Student:	**Duy Van Khanh Phan**
Study Programme:	B2647 Information and Communication Technology
Study Branch:	2612R025 Computer Science and Technology
Title:	Individual Professional Practice in the Company Absolvování individuální odborné praxe
The thesis language:	English

Description:

1. The student shall undergo individual practice in the company: Hofri s.r.o.
2. The final report structure shall be the following:
a/ Specification of the professional orientation of the firm where the student underwent his/her professional practice, specification of his/her occupational category.
b/ The list of tasks the student was assigned in the course of his/her professional practice including their time spans.
c/ Methods chosen when dealing with the given tasks.
d/ Theoretical and practical knowledge and skills acquired in the course of his/her professional practice.
e/ Knowledge and skills the student were missing in the course of his/her professional practice.
f/ Results achieved in the course of his/her professional practice and the general assessment of them.

References:

In accordance with the tutor guiding the student's professional practice.

Extent and terms of a thesis are specified in directions for its elaboration that are opened to the public on the web sites of the faculty.

Supervisor: **doc. Dr. Ing. Eduard Sojka**

Consultant: Ing. Jiří Hofrichter

Date of issue: 01.09.2018
Date of submission: 30.04.2019

doc. Ing. Jan Platoš, Ph.D.
Head of Department

prof. Ing. Pavel Brandštetter, CSc.
Dean

I hereby declare that this bachelor's thesis was written by myself. I have quoted all the references I have drawn upon.

Ostrava, April 30th 2019 .....

I hereby agree to the publishing of the bachelor's thesis as per s. 26, ss. 9 of the Study and Examination Regulations for Bachelor's Degree Programmes at VŠB – Technical University of Ostrava.

Ostrava, April 30th 2019 .....

I wish to express my appreciation for doc. Dr. Ing. Eduard Sojka for supervising and guiding me through this thesis. I also would like to thank Ing. Petr Lukáš for introducing me to the company Hofri s.r.o. and our colleagues in the company Hofri s.r.o.

Abstrakt

Bakalářská práce popisuje absolvování individuální odborné praxe ve firmě Hofri s.r.o. Cílem praxe je zmapovat současné cloudové aplikace společnosti, navrhnout a implementovat vývojové a produkční prostředí založené na kontejnerové technologii, navrhnout a implementovat řetězec pro kontinuální integraci a dodávání nových verzí aplikace. Práce obsahuje teoretické znalosti o kontejnerové technologii a praktické informace získané při výkonu mé praxe, její výsledek & dopad na firmu a na mě.

Klíčová slova: kontejnér, Docker, docker-compose, Amazon Web Services, kontinuální integrace a dodávání

Abstract

This bachelor thesis describes my individual professional practice in the company Hofri s.r.o. The goal of the practice is to containerize the company's current cloud applications, design and implement containerized development and production environment, continuous integration and delivery pipelines. The thesis contains theoretical knowledge about container technology, practical information how I performed my practice and the result & impact of the practice to the company and to myself.

Key Words: container, Docker, docker-compose, Amazon Web Services, continuous integration and delivery

Contents

List of symbols and abbreviations

API	–	Application programming interface
AWS	–	Amazon Web Services
CD	–	Continuous delivery
CI	–	Continuous integration
CLI	–	Command-line interface
DevOps	–	Development and operation
DI	–	Dependency injection
ECR	–	Elastic container registry
ECS	–	Elastic container service
IDE	–	Integrated development environment
OS	–	Operating system
PHP-FPM	–	PHP - Fast Process Manager
SSH	–	Secure Shell
UI	–	User interface

List of Figures

Listings

1 Introduction

I chose the option to perform individual professional practice at the company Hofri s.r.o. for my bachelor thesis. The goal of the practice is to containerize cloud applications using Docker, design and implement containerized development environment using docker-compose, production environment using Amazon Web Services, continuous integration and delivery pipelines using Bitbucket pipeline.

The web application of the company, contains several components (web server, script processor, source code ...), was natively installed on Linux servers, this faces multiple operation problems (to name a few: if a server is compromised with malicious software, it may go undetected for a while; when there is demand for new server, it takes long time for the new server to be ready; the current application's release new feature process takes long time), the company realized that containerized platform is the solution to these problems, therefore I was tasked to study container technology, get to know the company's application and containerize it, then I designed and implemented the environment to run the containerized application, after that, to ease the process of developing and releasing new application's features, I got familiar with the developing and releasing workflow and implemented integration pipeline so developers can test their changes and commit source code to version control system, and delivery pipeline to deploy new application version to production environment.

In the following parts of the thesis, first I describe the company Hofri s.r.o., its focus, vision and my position in the company, then I list out the tasks I performed during the practice and the timespan of each task, next I present theoretical knowledge related to container technology, after that I describe how I solved other practical tasks, the last part I mention the result of the practice and its impact to the company and to myself.

2 Company's and author's profile

2.1 About the company

Hofri s.r.o. was established in 2016 by Ing. Jiří Hofrichter[1], one of its projects is KVIKYMART[2], a solution for online spare parts retailer with connection to TecDoc catalog[3] that helps retailers to categorize their stocks from wholesaler, therefore enables end users to easily find the parts needed for their cars. As of March 2019, Hofri s.r.o. is having collaborations with more than 50 automotive aftermarket retailers from the Czech Republic, Slovakia, Poland, Hungary, Croatia and aiming to spread to other countries in the European Union and to the United States of America.

Tech stack using by the company can be found at StackShare[4], to name a few: main programming language on backend is PHP, TypeScript & Java; on frontend is JavaScript & React; PostgreSQL database; web server powered by Nginx and the applications are hosted on AWS.

2.2 About the author

I was working as a DevOps engineer at Hofri s.r.o., the role of this position is to take care of the operations during whole application's development cycle, i.e. plan, implement, test, release, deploy, operate, monitor and troubleshoot. I had the opportunity to experience programming language like PHP, TypeScript, JavaScript, Bash, Structured query language of MySQL and PostgreSQL, fulltext search engine Elasticsearch, in-memory data structure Redis, web server Nginx, version control system Git, Docker container and most significantly AWS.

After a time working as DevOps engineer with AWS, I developed the passion for DevOps career and wanted to be an AWS Certified DevOps Engineer[5].

[1] https://www.linkedin.com/in/hofrichterjiri/
[2] https://kvikymart.cz/
[3] a database containing detail of car parts and its manufacturers, vehicles and its motorization ...
[4] https://stackshare.io/kvikymart/kvikymart-platform
[5] https://aws.amazon.com/certification/certified-devops-engineer-professional/

3 List of assigned tasks

- **Understand the application of company and configuration of each component**

  Timespan: 20 hours

  The task is to figure out how the application is hosted on Linux servers, which components are running on the server and extract its configuration.

- **Study container technology**

  Timespan: 250 hours

  The task is to gather theoretical knowledge and best-practices in the field of container technology.

- **Containerize company's application**

  Timespan: 100 hours

  The task is to apply the theoretical knowledge in container technology to containerize each component of the company's application.

- **Design and implement containerized environment**

  Timespan: 100 hours

  The task is to orchestra the containers in the previous task in development and production environment.

- **Understand the developing and deploying workflow of the company**

  Timespan: 20 hours

  The task is to understand the process when the developers implement, test and release new features, how the features are deployed after committing to version control system.

- **Design and implement continuous integration and delivery pipelines**

  Timespan: 150 hours

  The task is to create an automated process to test and deploy new containers to production environment when the code is committed to version control system, partially using the information from previous task.

4 Container technology and container

Container technology utilizes OS-level virtualization for the development and operation of distributed applications without having to run a separate virtual machine for each application. Instead, these applications run in isolated environments (called containers) on a single host and share access to the core of the operating system. Since containers share the same OS kernel as a host, their operation is more efficient than running separate virtual machines. [1]

A container is a standard unit of software that packages up code and all its dependencies, therefore the application runs quickly and reliably from one computing environment to another.

Figure 1 compares the structure of application in containers and in virtual machine.

4.1 Docker

Docker is an open source project, firstly released in 2013, aiming to provide a unified interface for isolating application into containers in Linux and Windows. Docker consists of three components:

- **Engine:** The Docker daemon, named `dockerd`, is a persistent background process managing Docker containers and container objects. The Docker client program, named `docker`, provides CLI allowing users to interact with Docker daemon. Listing 1 shows some Docker CLI commands.

```
# To run a new container, if IMAGE_TAG is not specified, 'latest' tag will
    be assumed
docker run IMAGE_NAME[:IMAGE_TAG]
# To start a stopped container
docker start [CONTAINER_ID|CONTAINER_NAME]
# To list containers
docker ps
# To inspect a container
docker inspect [CONTAINER_ID|CONTAINER_NAME]
```

Listing 1: Sample Docker CLI commands

- **Objects:**
  - **Image** is a read-only template used to build containers. Images are built from instructions stored in Dockerfile (see §4.2). Built images are stored in registry.
  - **Container** is a runnable instance of an Docker image.
  - **Volume** to persist data when container is running.

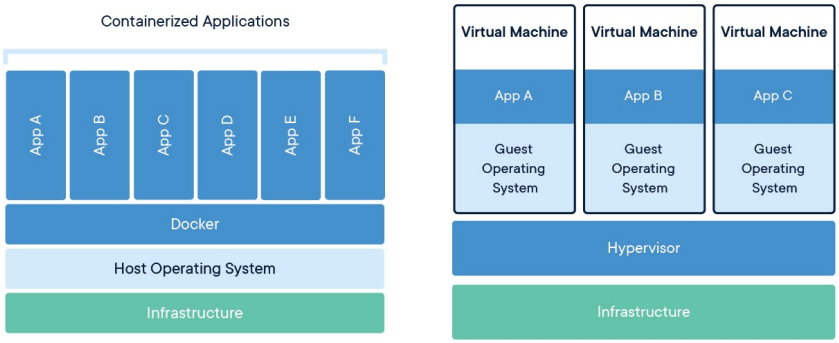

Figure 1: Compare structure of application in containers and in virtual machine [3]

- **Registries:** A Docker registry is a repository for Docker images. Docker registry can host both public repositories and private repositories. This thesis covers Docker Hub (see §4.4) and Amazon Elastic container registry (see §7.1).

4.2 Dockerfile

Dockerfile is a text document containing the instructions to build a Docker image. Is it not recommended to put credentials (e.g. API key, private SSH key ...) into Dockerfile. Listing 2 shows a sample Dockerfile aimed to build image for containers to run Nginx server with extra config and programs (`curl`, `vim`, `htop`).

```
# Specify base image
FROM nginx:1.15
# Install extra programs and remove cache so the image can be lightweight
RUN apt-get update && apt-get install -y \
    curl vim htop                        \
    && apt-get purge                     \
    && rm -rf /var/lib/apt/lists/*
# Copy config from host machine
COPY config /etc/
# Set working directory
WORKDIR /src
# Specify the listening port
EXPOSE 80
# Final command to run when container is run
CMD ["nginx", "-g", "daemon off;"]
```

Listing 2: Sample Dockerfile

4.3 Docker image

Docker image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings [2]. Listing 3 shows some sample CLI commands to work with Docker image.

```
# To build Docker image from Dockerfile
docker build -t IMAGE_NAME:IMAGE_TAG PATH_TO_DOCKER_FILE
docker build -t my-nginx:1 .
# To list local images
docker image ls
# To inspect an image
docker image inspect IMAGE_NAME:IMAGE_TAG
# To remove an image
docker image rm IMAGE_NAME:IMAGE_TAG
```

Listing 3: Sample CLI commands to work with Docker image

4.4 Docker Hub

Docker Hub[6] is a registry can host both public and private Docker image repositories. With free plan, individuals are allowed to have one private repository and unlimited public repositories. Listing 4 shows some sample CLI commands to work with Docker registry.

```
# To pull Docker image from Docker hub, if IMAGE_TAG is not specified, 'latest'
    tag will be assumed
docker pull IMAGE_NAME[:IMAGE_TAG]
docker pull nginx:1.15.9-alpine
# To push local image to repository
docker push IMAGE_NAME[:IMAGE_TAG]
# To login to private registry
docker login -u USERNAME -p PASSWORD
```

Listing 4: Sample CLI commands to work with Docker registry

[6] https://hub.docker.com/

5 Containerize company's application

To containerize company's application, I identified and write Dockerfile for each component:

5.1 Nginx

Nginx (pronounced "engine-x") is an open source reverse proxy server for HTTP, HTTPS, SMTP, POP3, and IMAP protocols, as well as a load balancer, HTTP cache, and a web server. [4]

```
# Specify base image
FROM nginx:1.15
# Logs from nginx base image are redirected to stdout, we remove the symlinks
    so logs can be store as files
RUN rm /var/log/nginx/*
# Install extra programs
RUN apt-get update && apt-get install -y \
    curl vim htop                        \
    && apt-get purge                     \
    && rm -rf /var/lib/apt/lists/*
# Copy config from host machine
COPY config /etc/
# Base nginx image already EXPOSE-d port 80 therefore we don't need to specify
    the listening port
# Final command to run when container is started
CMD ["nginx", "-g", "daemon off;"]
```

Listing 5: Dockerfile for nginx image

5.2 Vhost generator

The Nginx server hosts multiple domain names with different root directories, therefore a server block is needed for each domain name. Server block is a configuration block containing individual Nginx configuration for a domain name (e.g. which domain name the server is listening to, what is root directory, where to store logs . . .). In the company, server block is referred as vhost, short term for virtual host, an equivalent term in Apache server.

```
server {
    listen 80;
    server_name example.com www.example.com;
    index index.php;
```

```
    error_log /var/log/nginx/example.com-error.log;
    access_log /var/log/nginx/example.com-access.log;
    include common.d/php.conf;
    root /path/to/example.com/www;
}
```

Listing 6: Sample server block

I implemented a container to run a script that queries the list of domain name stored in database, generates server blocks and stores in individual vhost files. Theses vhost files are later included in the main Nginx configuration.

```
FROM node:11-alpine
COPY . /vhost-generator
WORKDIR /vhost-generator
RUN npm install
ENTRYPOINT ["node"]
CMD ["/vhost-generator/vhost-generator.js"]
```

Listing 7: Dockerfile for vhost generator image

The script for generating vhost files(`vhost-generator.js`) is written in JavaScript and can be found at Appendix A;

5.3 PHP-FPM

```
FROM php:7.1-fpm
RUN apt-get update && apt-get install --yes           \
    libpng-dev libxml2-dev libjpeg-dev libmcrypt-dev  \
    libzip-dev libyaml-dev libfreetype6-dev libxpm-dev \
    libicu-dev libxslt-dev libpq-dev libwebp-dev      \
    git parallel vim                                  \
&& docker-php-ext-configure gd                                \
    --with-jpeg-dir=/usr/include/ --with-xpm-dir=/usr/include/ \
    --with-webp-dir=/usr/include/ --with-freetype-dir=/usr/include/ \
&& docker-php-ext-install                \
    json pdo mysqli bcmath mbstring      \
    intl xml pgsql pdo_pgsql opcache gd \
    soap xsl pcntl pdo_mysql mcrypt      \
&& pecl install                          \
```

```
    redis-3.1.6 zip yaml                \
&& docker-php-ext-enable                \
    redis zip yaml                      \
&& apt-get purge                        \
&& rm -rf /var/lib/apt/lists/*
COPY config /usr/local/etc/
CMD ["php-fpm"]
WORKDIR /mono
```

Listing 8: Dockerfile for PHP-FPM image

5.4 Filebeat

Filebeat is an application from the ElasticStack[7] that delivers and centralizes Nginx log from multiple containers to ElasticSearch to ease log analytics.

```
FROM elastic/filebeat:6.3.0
COPY --chown=filebeat filebeat.yml /usr/share/filebeat/filebeat.yml
RUN chmod 0644 /usr/share/filebeat/filebeat.yml
```

Listing 9: Dockerfile for Filebeat image

5.5 The company's application source code

The PHP scripts in the company's application source code need to be analyzed to create DI containers before deployment. To create DI containers, the environment needs extra dependencies and libraries, but the final Docker image doesn't need to include those, therefore I utilized multi-stage build[8] to build a final lightweight image that contains only source code and DI containers.

```
# Stage one
FROM node-php:37_20d82e5 AS stage-one
COPY . /mono
WORKDIR /mono
RUN ls -1 Eshops/ | parallel -j 8 'php Eshops/{}/www/index.php oops:morozko:
    warmup'

# Stage two
```

[7] https://www.elastic.co/products/
[8] https://docs.docker.com/develop/develop-images/multistage-build/

```
FROM busybox
COPY --chown=33:33 --from=stage-one /mono /mono
VOLUME /mono
```

Listing 10: Dockerfile for application's source code image

6 Design and implement containerized development environment

To orchestra containers at development environment, I use `docker-compose`.

6.1 docker-compose

`docker-compose` is a tool from Docker software to define and run multi-container application. A YAML file contains configurations of how containers should start, which ports should be mapped to host machine, which directory on host machine should be mounted to containers ...

```
version: '3' # This is version of docker-compose syntax

services:
    nginx: # Web server container
        container_name: nginx
        image: nginx:1.15 # Container can be started from Docker image
        ports:
            - 80:80 # port-on-host-machine:port-on-container
        volumes:
            # directory-on-host-machine:directory-on-container
            - ./nginx/nginx.conf:/etc/nginx/nginx.conf
            - ../mono:/mono

    php: # Script processor container
        container_name: php
        build: ./php # Container can be also started from Dockerfile stored in
            this directory
        volumes:
            - ../mono:/mono
            - ~/.aws:/root/.aws
            - ~/.aws:/var/www/.aws
        environment: # container's environment variable
            - DOCKER_MONO_DEV_RUNNING=true

    mysql: # Database container
        container_name: mysql
        image: mysql:5.7
        volumes:
```

```
            - ../mono/Core/sql:/docker-entrypoint-initdb.d/
        ports:
            - 3306:3306
        environment:
            - MYSQL_ROOT_PASSWORD=root_password
            - MYSQL_DATABASE=default_database_name

    redis: # In-memory data storing container
        container_name: redis
        ports:
            - 6379:6379
        image: redis:5.0

    phpredmin: # UI to manage data in Redis
        container_name: phpredmin
        image: faktiva/php-redis-admin
        ports:
            - 8089:80
        volumes:
            - ./phpredmin/config.php:/var/www/html/php-redis-admin/app/config/
              config.php

    adminer: # Database client
        container_name: adminer
        image: adminer:4.6
        ports:
            - 8080:8080
```

Listing 11: `docker-compose.yml` file to orchestra containers at development environment

docker-compose also comes with CLI to controll containers described in `docker-compose.yaml`.

```
# To spin up all containers specified in docker-compose.yaml
docker-compose up
# To spin up only selected containers
docker-compose up nginx php redis
```

Listing 12: Sample CLI commands to work with docker-compose

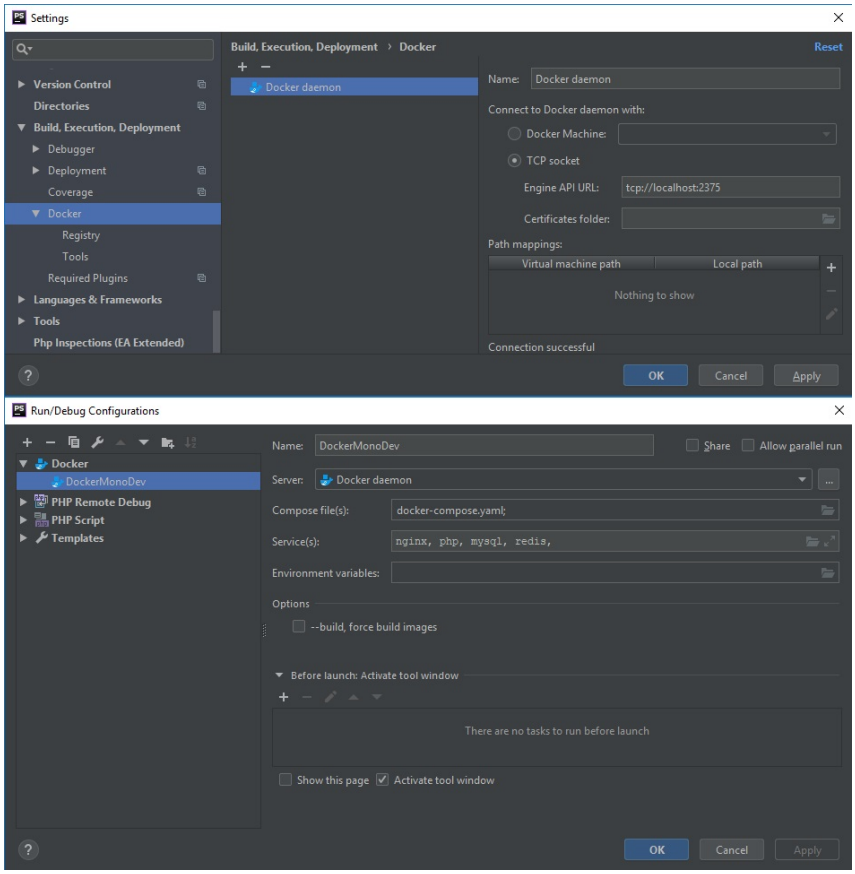

Figure 2: Integrate docker-compose with PhpStorm

6.2 Integrate with PhpStorm IDE

Developers at Hofri s.r.o. use PhpStorm[9], therefore I instruct them how to interact with docker-compose from PhpStorm. See fig. 2.

- In Settings | Build, Execution, Deployment | Docker: Connect with Docker daemon at `tcp://localhost:2375`.
- Create new Docker | docker-compose run/debug configuration, select the previous Docker daemon as server, specify the `docker-compose.yaml` file and select all containers.

[9] An IDE for PHP from JetBrains, https://www.jetbrains.com/phpstorm/

7 Design and implement containerized production environment

7.1 Amazon Elastic container registry

Amazon ECR is an Amazon fully-managed Docker container registry that offers unlimited private repositories with the cost of $0.10 per GB-month.

Listing 13 shows some sample CLI commands to work with Docker registry.

```
# To retrieve the login command to use to authenticate Docker client to
    registry
aws ecr get-login --no-include-email --region REGION
# To retrieve login command and execute it
    # for Linux
    aws ecr get-login --no-include-email --region REGION | sh
    # for Windows command prompt
    aws ecr get-login --no-include-email --region REGION | cmd
    # for Windows Powershell
    aws ecr get-login --no-include-email --region REGION | powershell
# To pull image from ECR
docker pull AWS_ID.dkr.ecr.REGION.amazonaws.com/IMAGE_NAME:IMAGE_TAG
# To push local image to ECR
docker push AWS_ID.dkr.ecr.REGION.amazonaws.com/IMAGE_NAME:IMAGE_TAG
```

Listing 13: Sample CLI commands to work with Amazon Elastic container registry

7.2 Amazon Elastic container service

Amazon Elastic container service is a Docker container orchestration service that allow running and scaling easily containerized applications on AWS. The following list mentioned some terms using in AWS containerized infrastructure.

- **Task** is a group of different containers needed to perform one job. In my practice, a task consists of web server, vhost files generator, script processor, log collector and source code container.
- **Task definition** is a document required to run a task. Some example parameters can be specify in task definition: the Docker image to use for each container, the computing resource needed for the task, the command the container will run when it's started ...
- **Service** is a group of specified number of task. If a task fails or stops unexpectedly, ECS will launch another task predefined by the task definition to replace it and maintain the

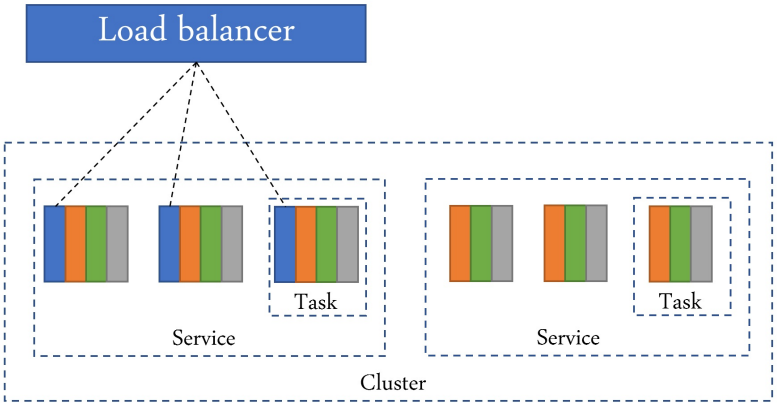

Figure 3: Sample simplified AWS containerized infrastructure

desired count of tasks in the service. The service is also capable of scaling when there is high load/usage as we define, i.e. increase or decrease the desired count of tasks in the service.

- **Cluster** is a group of different services.
- **Load balancer** automatically distributes incoming traffic across multiple targets. In my practice, the load balancer balances the traffic to web server container.

Figure 3 depicts simplified AWS infrastructure hosting 2 containerized applications with one application attached to a load balancer.

In this task I created and configured AWS Load balancer, created AWS ECR repository for each Docker image, created Task definition for the application, created and configures AWS ECS cluster and services.

8 Design and implement continuous integration and delivery pipelines

8.1 Bitbucket and Bitbucket pipeline

Bitbucket[10] is a Git solution owned by Atlassian[11]. With free plan for individuals and small teams with up to 5 users, Bitbucket offers unlimited public and private git repositories and 50 build minutes for pipeline per month. With academic subscription, users can have up to 500 build minutes free per month.

Bitbucket pipeline is series of Docker container, called "step", running sequentially or parallelly, inside each step is a list of bash command specified in `bitbucket-pipelines.yml`. A pipeline can only succeed when every bash command succeeds, i.e. command exits with code 0. Bitbucket pipeline can be triggered manually, by git push signal or when pull requests are created or merged. Listing 14 shows example of a pipeline step that login to Docker Hub registry, build and push image.

```
- step:
    script:
        - docker login -u $DOCKER_USERNAME -p $DOCKER_PASSWORD
        - docker build -t $IMAGE_NAME:$IMAGE_TAG .
        - docker push $IMAGE_NAME:$IMAGE_TAG
```

Listing 14: Sample step in `bitbucket-pipelines.yml`

8.2 Continuous integration pipeline

Continuous integration is the practice of early, frequently merging code changes into the main branch so developers can detect errors quickly and locate them easily.

I implemented CI pipeline that automatically runs when pull request is created, and when there is new commit to the pull request. When the pipeline runs, target and destination branch are virtually merged, thus "integration", then static analysis for PHP scripts and JavaScript scripts run parallelly. The code quality check for JavaScript output warnings (if any) directly to console and exit with code 0, therefore the pipeline succeeds and developers aren't aware of the warnings unless they open the Bitbucket pipeline UI, so I write bash commands to send the warnings directly to the developers through Slack[12].

The content of `bitbucket-pipelines.yml` for CI pipeline can be found at Appendix B.

[10] https://bitbucket.org/
[11] https://atlassian.com/
[12] A messaging program, https://slack.com/

8.3 Continuous delivery pipeline

Continuous delivery is an extension of continuous integration, that means the release and deploy process follows automated test. The pipeline needs to do the following steps and the time each step averagely took:

- Install libraries — 1 min.
- Compile common resources — 3 min. 30 sec.
- Compile individual resources for each client — 14 min. 30 sec.
- Build Docker image and push to ECR — 5 min. 45 sec.
- Deploy — 1 min.

The whole pipeline on average takes 26 min. to build, I analyzed & recognized the step "Compile individual resources for each client" is parallelizable, therefore the new version of pipeline on average only takes 12 min. to build.

The content of `bitbucket-pipelines.yml` for CD pipeline can be found at Appendix C. The script of parallelizing "Compile individual resources for each client" is written in Bash and can be found at Appendix D.

9 Result evaluation

The outcome of the practice is a huge success for both parties.

As for the company Hofri s.r.o., they acquired:

- A local containerized development environment blueprint that saves new employees the hassle of installing libraries, dependencies and configurations on new devices — this increases employees' performance and reduces company's expense.
- An elastic containerized hosting platform that is capable of rapidly scaling in and out according to their clients' load, enable Infrastructure as Code and Disaster recovery — these things didn't exist before the practice.
- Pipelines that ease the process of approving - staging - releasing new features — used to take more than 45 minutes before the practice, now only takes 12 minutes.

As for the author, I acquired work experience and ethic that cannot be taught in any educational institution, I had opportunities to work in team, to work under pressure, to practice critical thinking and constructive criticism, and most importantly, I found the passion in working as a DevOps engineer with AWS, in which I want to pursue a career.

References

[1] KAČÍK, Jakub. Virtualizační platformy, kontejnerové technologie a Cloud služby [online]. Ostrava, 2018 [cit. 2019-03-01]. Dostupné z: http://hdl.handle.net/10084/128640. Bakalářská práce. Vysoká škola báňská - Technická univerzita Ostrava.

[2] What is Container: A standardized unit of software [online]. [cit. 2019-03-01]. Available at: https://docker.com/what-container

[3] Containers and Virtual Machines Together [online]. [cit. 2019-03-01]. Available at: https://docker.com/what-container

[4] Nginx official docker image repository [online]. [cit. 2019-03-01]. Available at: https://hub.docker.com/_/nginx

A vhost-generator.js for generating vhost files

```
const IS_DEV_ENV = !process.env.AWS_EXECUTION_ENV;
const
    SELECT_ALL_QUERY = 'SELECT basename FROM basename',
    GENERATED_VHOST_DIR = IS_DEV_ENV ? 'generated-vhost/' : '/etc/nginx/
        sites-enabled/',
    SECRET_ID = '/secretId',

    DATABASE_NAME = process.env.DATABASE_NAME,
    DEV_DB_USER = process.env.DATABASE_USER,
    DEV_DB_PASSWORD = process.env.DATABASE_PASSWORD,
    DEV_DB_HOST = process.env.DATABASE_HOST;
const
    pg = require('pg'),
    fs = require('fs'),
    aws = require('aws-sdk'),
    path = require('path');

aws.config.apiVersions = {secretsmanager: '2017-10-17'};
aws.config.update({region: 'eu-west-1'});

const secretsManagerClient = new aws.SecretsManager();

function getSecret(secretId) {
    return secretsManagerClient.getSecretValue({SecretId: secretId}).
        promise();
}
function generateVhost(item, index, array) {
    try {
        const basename = item['basename'];
        let vhostConfig = '
        server {
            listen 81;
            server_name ${basename};
            index index.php index.html index.htm;

            error_log /var/log/nginx/${basename}-error.log;
            access_log /var/log/nginx/${basename}-access.log alb;
```

```
            root /mono/Eshops/${basename}/www;
        }
        ';
        fs.writeFileSync(path.join(GENERATED_VHOST_DIR, '${basename}.vhost')
            , vhostConfig);
    } catch (e) {
        console.log(item, e);
    }
}
async function main() {
    let dbConfig;

    if (IS_DEV_ENV) {
    dbConfig = {
        user: DEV_DB_USER,
        host: DEV_DB_HOST,
        database: DATABASE_NAME,
        password: DEV_DB_PASSWORD,
    };
    } else {
        const data = await getSecret(SECRET_ID);
        const secretData = JSON.parse(data.SecretString);

        dbConfig = {
            user: secretData.username,
            host: secretData.host,
            database: DATABASE_NAME,
            password: secretData.password,
            port: secretData.port,
        };
    }

    const pgClient = new pg.Client(dbConfig);

    await pgClient.connect();
    const items = await pgClient.query(SELECT_ALL_QUERY);
    await pgClient.end();
```

```
    items.rows.forEach(generateVhost);
}
main();
```

Listing 15: Script to generate vhost files

B bitbucket-pipelines.yaml for CI pipeline

```
- step:
    name: Composer install Core
    image: composer:1.8
    caches:
        - cache-composer-core
    script:
        - composer install --optimize-autoloader --ignore-platform-reqs --no-
            scripts --no-interaction --no-progress -d Core/
    artifacts:
        - Core/**
- parallel:
    - step:
        name: Test PHP code
        script:
            - cd Core
            - php -d memory_limit=2G vendor/bin/phpstan analyse src --level=0 -c
                phpstan.neon --no-progress
            - cd ../
            - mkdir --parents Data/Temp Data/Logs
            - ls -1 Eshops/ | parallel -j 8 'php Eshops/{}/www/index.php oops:
                morozko:warmup || php Eshops/{}/www/index.php oops:morozko:
                warmup -vvv'
    - step:
        name: Test JS code
        caches:
            - cache-node
        script:
            - cd Core/
            - npm install --production
            - LINT_OUTPUT=lint-output.txt
            - npm run lint -- -o $LINT_OUTPUT && if [ -f $LINT_OUTPUT ]; then
                cat $LINT_OUTPUT; fi || npm run lint
            - if [ -f $LINT_OUTPUT ]; then echo -e "\non branch
                $BITBUCKET_BRANCH" >> $LINT_OUTPUT; curl -X POST -H --silent --
                data "payload={\"text\":\"\`\`\`\n$(cat $LINT_OUTPUT)\n\`\`\`\"}
                " $SLACK_HOOK; fi
            - npm run test
```

Listing 16: bitbucket-pipelines.yaml for CI pipeline

C bitbucket-pipelines.yaml for CD pipeline

```
- step:
    name: Composer install
    image: composer:1.8
    caches:
        - cache-composer-core
        - cache-composer-commandlinetool
    script:
        - composer install --no-dev --optimize-autoloader --ignore-platform-
            reqs --no-scripts --no-interaction --no-progress -d Core/
        - composer install --no-dev --optimize-autoloader --ignore-platform-
            reqs --no-scripts --no-interaction --no-progress -d CommandLineTool/
    artifacts:
        - Core/**
        - CommandLineTool/**
- step:
    name: Core resources
    caches:
        - cache-node
    script:
        - npm install --production --prefix Core/
        - npm run build --prefix Core/
        - aws s3 cp Core/resources s3://bucketname/core --recursive --acl
            public-read --only-show-errors
    artifacts:
        - Core/resources/build/*/manifest.json
- parallel:
    - step:
        name: Eshop resources
        caches:
            - cache-node
        script:
            - chmod +x ./DeployScripts/compile-and-upload-resources.sh
            - ./DeployScripts/compile-and-upload-resources.sh
        artifacts:
            - Eshops/*/www/build/manifest.json
    - step:
        name: Eshop resources
```

```
        caches:
            - cache-node
        script:
            - chmod +x ./DeployScripts/compile-and-upload-resources.sh
            - ./DeployScripts/compile-and-upload-resources.sh
        artifacts:
            - Eshops/*/www/build/manifest.json
    - step:
        name: Eshop resources
        caches:
            - cache-node
        script:
            - chmod +x ./DeployScripts/compile-and-upload-resources.sh
            - ./DeployScripts/compile-and-upload-resources.sh
        artifacts:
            - Eshops/*/www/build/manifest.json
- step:
    name: Push image to ECR and deploy to Test
    image: atlassian/pipelines-awscli
    deployment: staging
    services:
        - docker
    script:
        - FILEPATH=Core/resources/build/legacy/manifest.json && if [ ! -f
          $FILEPATH ]; then echo Cannot find $FILEPATH && exit 1; fi
        - FILEPATH=Core/resources/build/react/manifest.json && if [ ! -f
          $FILEPATH ]; then echo Cannot find $FILEPATH && exit 1; fi
        - apk add parallel
        - ls -1 Eshops/ | parallel -j 8 'FILEPATH=Eshops/{}/www/build/manifest.
          json && if [ ! -f $FILEPATH ]; then echo Cannot find $FILEPATH &&
          exit 1; fi'
        - mkdir healthcheck && echo "KVIKYMART" > healthcheck/healthcheck
        - eval $(aws ecr get-login --region ${AWS_DEFAULT_REGION} --no-include-
          email)
        - BUILD_ID=${BITBUCKET_BUILD_NUMBER}_${BITBUCKET_COMMIT:0:7}
        - docker build --tag ${AWS_REGISTRY_URL}:${BUILD_ID} --build-arg
          AWS_ACCESS_KEY_ID=${AWS_ACCESS_KEY_ID} --build-arg
          AWS_SECRET_ACCESS_KEY=${AWS_SECRET_ACCESS_KEY} --build-arg
          AWS_DEFAULT_REGION=${AWS_DEFAULT_REGION} .
```

```
        - docker push ${AWS_REGISTRY_URL}:${BUILD_ID}
        - aws lambda invoke --function-name $DEPLOY_CONTROLLER --invocation-
            type RequestResponse --payload '{"clusterName":"'"$TEST_CLUSTER"'",
            "serviceName":"'"$TEST_SERVICE"'", "registryUrl":"'"
            $AWS_REGISTRY_URL"'", "buildId":"'"$BUILD_ID"'"}' --log-type Tail -
            | grep "LogResult"| awk -F'"' '{print $4}' | base64 -d
- step:
    name: Deploy to Production
        image: atlassian/pipelines-awscli
        deployment: production
        trigger: manual
        script:
            - BUILD_ID=${BITBUCKET_BUILD_NUMBER}_${BITBUCKET_COMMIT:0:7}
            - aws lambda invoke --function-name $DEPLOY_CONTROLLER --invocation-
                type RequestResponse --payload '{"registryUrl":"'"
                $AWS_REGISTRY_URL"'", "buildId":"'"$BUILD_ID"'"}' --log-type
                Tail - | grep "LogResult"| awk -F'"' '{print $4}' | base64 -d
```

Listing 17: bitbucket-pipelines.yaml for CD pipeline

D Script to parallelize "Compile individual resources for each client"

```
#!/bin/bash

currentThreadId=$BITBUCKET_PARALLEL_STEP;
totalThreadCount=$BITBUCKET_PARALLEL_STEP_COUNT;
lastThreadId=$((totalThreadCount - 1));

eshopArray=($(ls -1 Eshops/));
eshopArrayCount=${#eshopArray[@]};

eshopToProcessPerThreadCount=$((eshopArrayCount / totalThreadCount));

fromIndex=$((currentThreadId * eshopToProcessPerThreadCount));

if [[ ${currentThreadId} -eq ${lastThreadId} ]]
then
    eshopToProcessArray=${eshopArray[@]:fromIndex};
else
    eshopToProcessArray=${eshopArray[@]:fromIndex:eshopToProcessPerThreadCount
        };
fi

parallel -j 8 'echo {} && npm install --prefix Eshops/{} && npm run build --
    prefix Eshops/{} || echo Error in {}' ::: ${eshopToProcessArray};
parallel -j 8 'aws s3 cp Eshops/{}/www s3://bucket-name/{} --recursive --acl
    public-read --exclude "index.php" --only-show-errors' ::: ${
    eshopToProcessArray};
```

Listing 18: Script to parallelize "Compile individual resources for each client"